

# 32 位微控制器

## HC32M120 系列的串行外设接口 SPI

适用对象

系列	产品型号
<b>HC32M120</b>	HC32M120J6TB
	HC32M120F6TB

# 目 录

<b>1</b>	<b>摘要.....</b>	<b>3</b>
<b>2</b>	<b>SPI 简介.....</b>	<b>3</b>
2.1	功能简介.....	3
2.2	主要特性.....	4
2.3	引脚配置.....	5
<b>3</b>	<b>SPI 通信说明 .....</b>	<b>6</b>
3.1	SPI 引脚分组 .....	6
3.2	数据格式.....	7
3.3	波特率.....	7
3.4	标准模式.....	8
<b>4</b>	<b>SPI 应用 .....</b>	<b>10</b>
4.1	主机模式.....	10
4.1.1	主机 4 线模式 .....	11
4.1.2	主机 3 线模式 .....	11
4.1.3	程序示例.....	12
4.2	从机模式.....	15
4.2.1	从机 4 线模式 .....	15
4.2.2	从机 3 线模式 .....	15
4.2.3	程序示例.....	15
<b>5</b>	<b>总结.....</b>	<b>18</b>
<b>6</b>	<b>版本信息 &amp; 联系方式.....</b>	<b>19</b>

## 1 摘要

本应用笔记主要介绍 HC32M120 系列 MCU 的串行外设接口（以下简称 SPI）的特点及使用方法，包括主从机模式、三线模式和四线模式的配置和应用。

## 2 SPI 简介

### 2.1 功能简介

HC32M120 系列 MCU 内部搭载一个通道的 SPI（系统框图如图 2-1），支持高速全双工串行同步传输。用户可根据需要进行三线/四线，主机/从机以及波特率的设置。

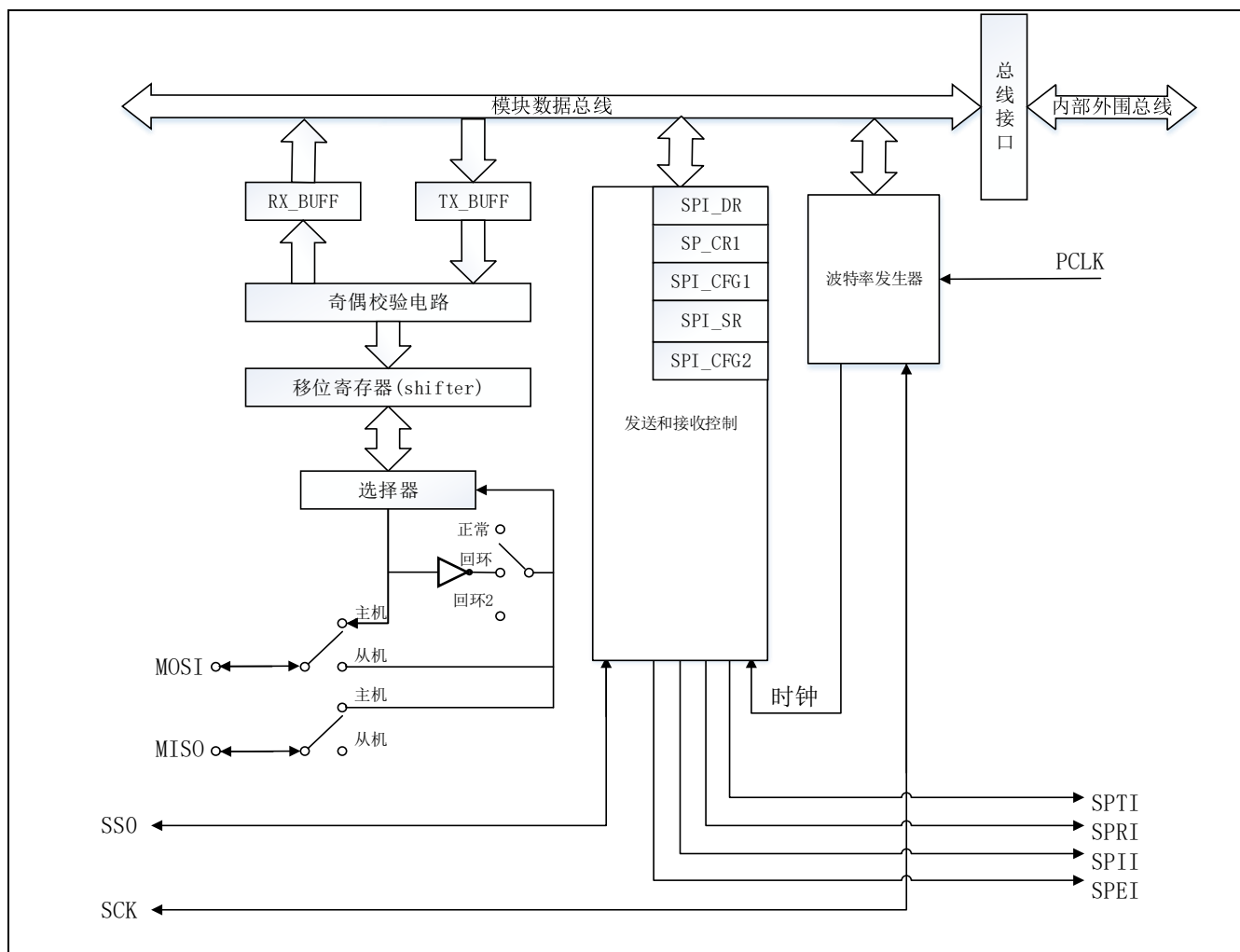


图 2-1 SPI 系统框图

## 2.2 主要特性

HC32M120 系列 MCU 的 SPI 的主要特性如表 2-1 所示：

要点	描述
通道数	1通道
串行通信功能	<ul style="list-style-type: none"> <li>支持4线式SPI模式和3线式时钟同步运行模式</li> <li>支持全双工和只传送两种通信方式</li> <li>可调整通信时钟SCK的极性和相位</li> </ul>
数据格式	<ul style="list-style-type: none"> <li>可选择数据移位顺序:MSB开始/LSB开始</li> <li>可选择数据宽度: 8/16位</li> <li>单次最多可传送或接收1帧宽度为16位的数据</li> </ul>
波特率	<ul style="list-style-type: none"> <li>主机模式下可通过内置专用波特率发生器对波特率进行调整，波特率范围为PCLK的2分频~256分频</li> <li>从机模式下允许的最大波特率为PCLK的6分频</li> </ul>
错误监测	<ul style="list-style-type: none"> <li>模式故障错误监测</li> <li>数据过载错误监测</li> <li>数据欠载错误监测</li> <li>奇偶校验错误监测</li> </ul>
片选信号控制	<ul style="list-style-type: none"> <li>每个通道配置一根片选信号线</li> <li>极性可调</li> </ul>
主机模式下的传输控制	<ul style="list-style-type: none"> <li>通过将数据写入数据寄存器启动传输</li> </ul>
中断/AOS源	<ul style="list-style-type: none"> <li>接收数据区域已满</li> <li>发送数据区域已空</li> <li>SPI错误（模式/过载/欠载/奇偶校验）</li> <li>SPI闲置</li> <li>传输完成</li> </ul>
低功耗控制	<ul style="list-style-type: none"> <li>可设置模块停止</li> </ul>
其他功能	<ul style="list-style-type: none"> <li>SPI初始化功能</li> </ul>

表 2-1 SPI 的主要特性

## 2.3 引脚配置

HC32M120 系列 MCU 的 SPI 的管脚配置如表 2-2 所示：

管脚名	端口方向	功能
SCK	输入/输出	通信时钟管脚
MOSI	输入/输出	主机数据传输管脚
MISO	输入/输出	从机数据传输管脚
SS0（或NSS）	输入/输出	从机选择输入/输出管脚

表 2-2 SPI 引脚配置

## 3 SPI 通信说明

### 3.1 SPI 引脚分组

HC32M120 系列 MCU 的 SPI 模块最多有两个引脚分组 A 和 B，引脚对应关系如表 3-1。

SPI 引脚分组	对应引脚	
SPI_PIN_GROUP_A	SPI_NSS_A	P13
	SPI_SCK_A	P14
	SPI_MOSI_A	P11
	SPI_MISO_A	P12
SPI_PIN_GROUP_B	SPI_NSS_B	P22
	SPI_SCK_B	P23
	SPI_MOSI_B	P20
	SPI_MISO_B	P21

表 3-1 SPI 引脚分组

## 3.2 数据格式

SPI 数据的位宽可配置位 8 位或 16 位，首先传输的位可配置为 LSB 或 MSB，支持奇偶校验。寄存器 SPI\_CFG2.DSIZE 决定数据的位宽，SPI\_CR1.PAE 决定数据最末位（如果首先传输的位是数据的 LSB，那么最末位就是数据的 MSB，反之则是 LSB）是奇偶校验位（SPI\_CR1.PAE=1）还是数据位（SPI\_CR1.PAE=0）。数据格式如图 3-1。

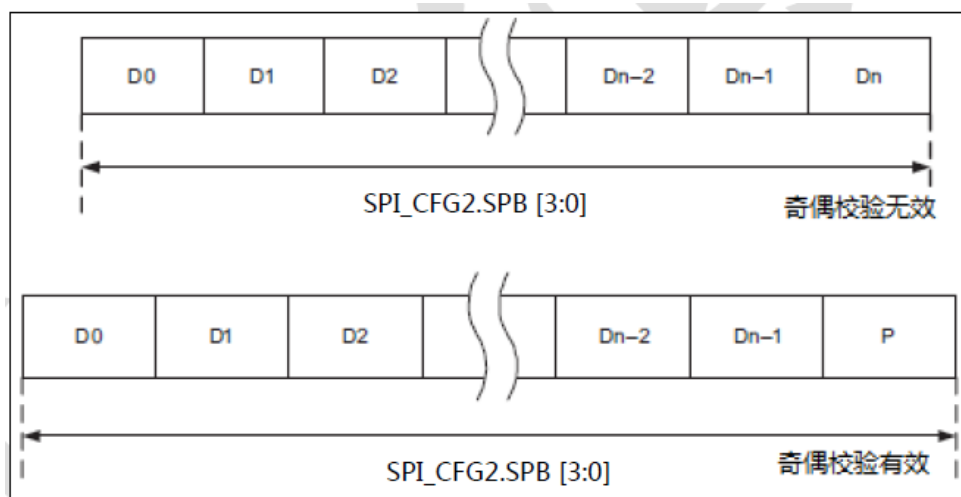


图 3-1 SPI 数据格式

## 3.3 波特率

主机模式下，SPI 时钟由内部波特率发生器提供，最大波特率为 PCLK/2；从机模式下，时钟为 SCK 脚输入，最大波特率为 PCLK/6（即输入的 SCK 的最高频率为 PCLK/6）。波特率取决于 SPI\_CFG2.MBR[2:0]位的设置。其计算公式如图 3-2，其中 N 为 MBR[2:0]位的设定值，范围是 0~7。

$$\text{波特率} = \frac{fpck}{2^{N+1}}$$

图 3-2 SPI 波特率计算公式

### 3.4 标准模式

HC32M120 系列 MCU 的 SPI 模块支持标准的 4 种 SPI 模式，如表 3-2 及图 3-3 和图 3-4。

SPI 模式	CPOL	CPHA	描述
模式 0	0	0	SCK 空闲时为低电平，数据采样在 SCK 奇数边沿，数据变化在 SCK 偶数边沿
模式 1	0	1	SCK 空闲时为低电平，数据采样在 SCK 偶数边沿，数据变化在 SCK 奇数边沿
模式 2	1	0	SCK 空闲时为高电平，数据采样在 SCK 奇数边沿，数据变化在 SCK 偶数边沿
模式 3	1	1	SCK 空闲时为高电平，数据采样在 SCK 偶数边沿，数据变化在 SCK 奇数边沿

表 3-2 SPI 模式描述

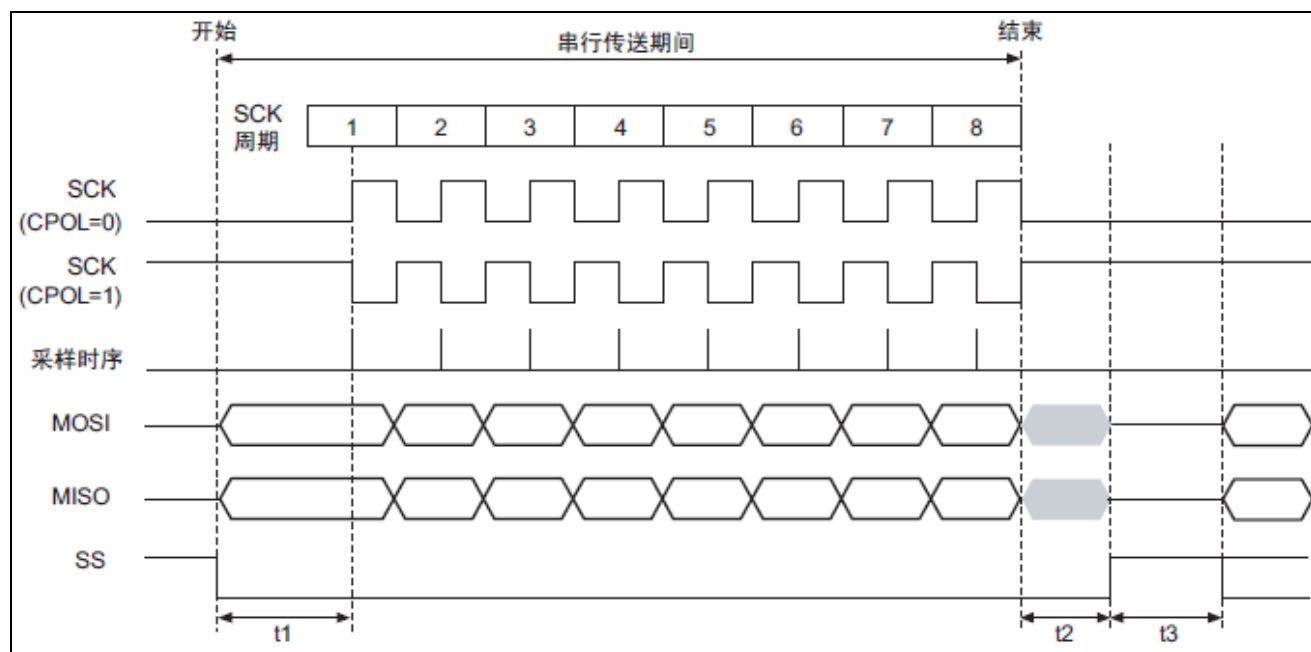


图 3-3 SPI 时序图（CPHA=0）



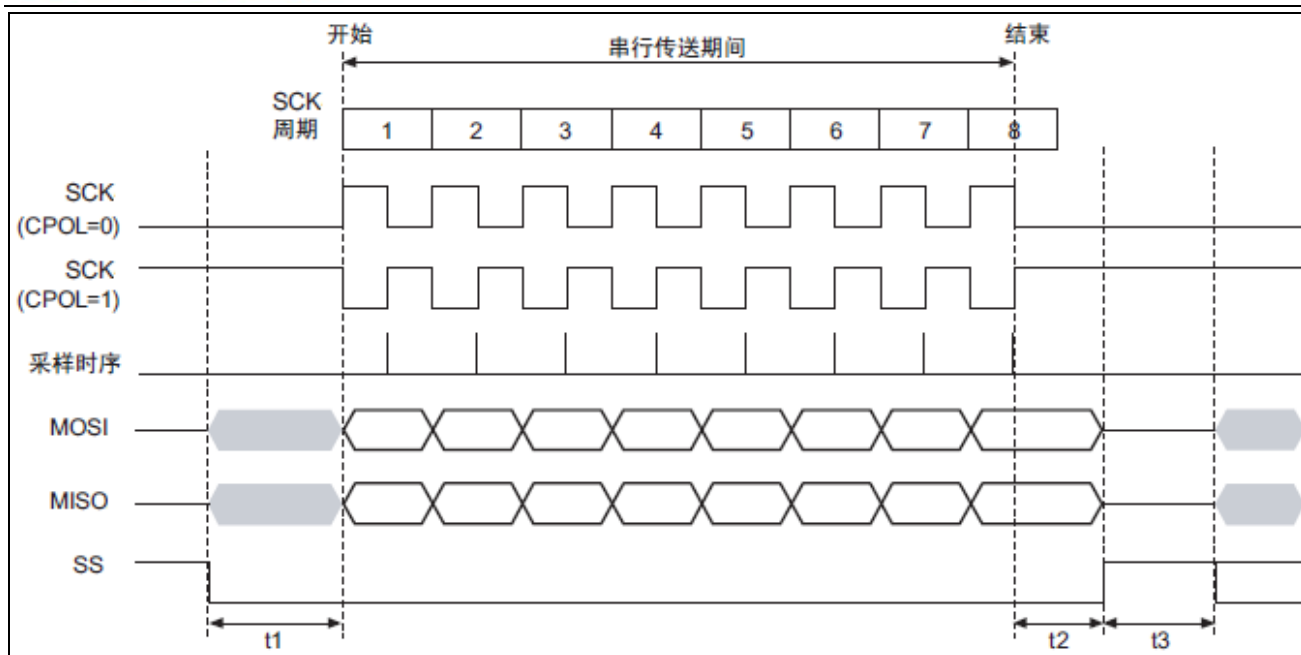


图 3-4 SPI 时序图 ( $CPHA=1$ )

## 4 SPI 应用

HC32M120 系列 MCU 的 SPI 模块支持 4 线模式和 3 线模式（时钟同步模式），每个运行模式下都可作为主机或从机进行串行通信。

### 4.1 主机模式

SPI 工作在主机模式下时，各管脚状态如表 4-1 所示。

模式		管脚名	管脚状态(PFS.ODS=0)	管脚状态(PFS.ODS=1)
SPI动作 (SPIMDS=0)	主机模式 (MSTR=1、 MODFE=0)	SCK	CMOS输出	OD输出
		SS0	CMOS输出	OD输出
		MOSI	CMOS输出	OD输出
		MISO	输入	输入
时钟同步运行 (SPIMDS=1)	主机模式 (MSTR=1)	SCK	CMOS输出	OD输出
		SS0（不使用）	Hi-Z（可作为通用I/O）	Hi-Z（可作为通用I/O）
		MOSI	CMOS输出	OD输出
		MISO	输入	输入

表 4-1 主机模式时 SPI 管脚状态

### 4.1.1 主机 4 线模式

在该模式下，主机在一个片选有效期之内，只发送或接收一个数据（8 位或 16 位，由 SPI\_CFG2.DSIZE 决定）。该模式下，SPI 所有引脚的电平都由 SPI 模块控制，时序如图 4-1，其中，通道 1 是 NSS，通道 2 是 SCK，通道 3 是 MOSI，通道 4 是 MISO，下同。

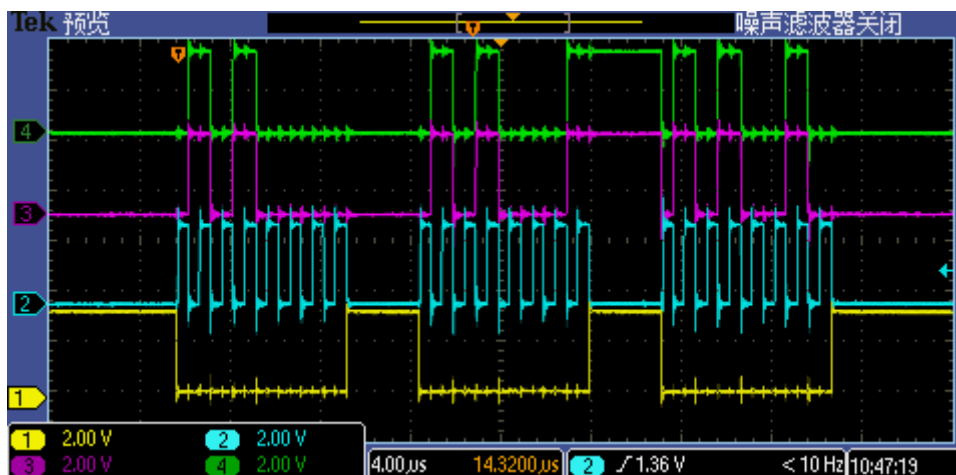


图 4-1 主机 4 线模式时序图

### 4.1.2 主机 3 线模式

该模式，相对主机 4 线模式，少了一个片选引脚，时序如图 4-2。

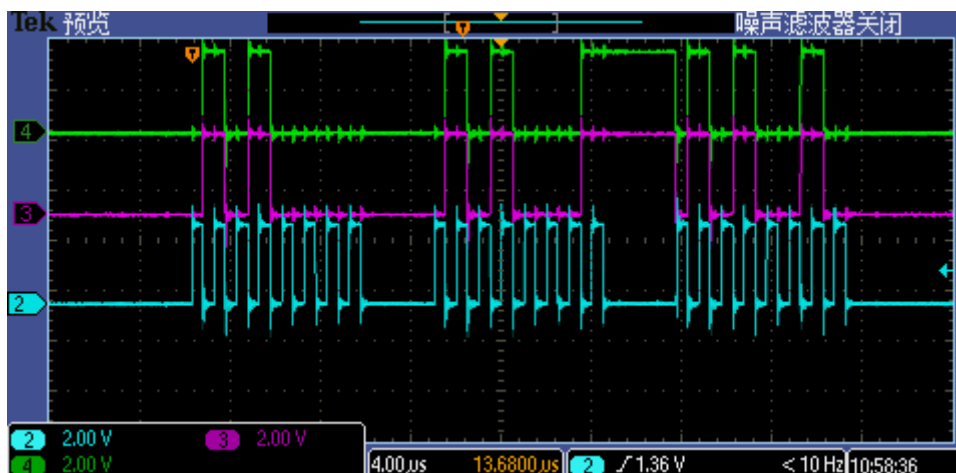


图 4-2 主机 3 线模式时序图

该模式可搭配一个自定义的片选引脚，实现在一个片选有效期内，发送或接收若干数据的应用，这也是 SPI 常见的应用，通信时序如图 4-3。

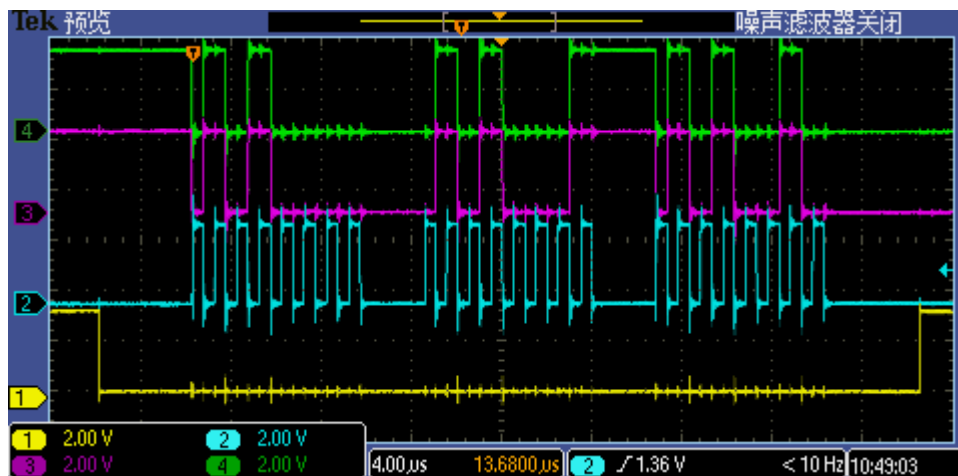


图 4-3 主机 3 线模式搭配自定义 NSS 引脚

### 4.1.3 程序示例

SPI 主机模式的各种配置和用法在例程 `spi_master_base` 中都有展示，其运行情况也如图 4-1、图 4-2 和图 4-3 所示。下面主要介绍比较关键的代码。

SPI 例程基本结构如程序清单 4-1 所示。

```
int32_t main(void)
{
    /* Configure the system clock to HRC32MHz. */
    SystemClockConfig();

    /* Configures SPI. */
    SpiConfig();

    /****** Configuration end, application start *****/

    while (1u)
    {
#ifdef SLAVE_TEST
        /* Write data to the slave. */
        m_au8SpiTxBuf[0u] = SPI_WRITE_SLAVE;
        SPI_Transmit((uint8_t *)&m_au8SpiTxBuf[0u], SPI_BUFFER_LENGTH);
        /* Delay for slave handling data. */
        DDL_Delay1ms(2u);

        /* Read data from the slave. */
        m_au8SpiTxBuf[0u] = SPI_READ_SLAVE;
        SPI_Transmit((uint8_t *)&m_au8SpiTxBuf[0u], SPI_BUFFER_LENGTH);
        /* Delay for slave handling data. */
        DDL_Delay1ms(2u);
#endif
    }
}
```

```

    SPI_Receive((uint8_t *)&m_au8SpiRxBuf[0u], SPI_BUFFER_LENGTH);
    // TODO: Use data received from the slave. Valid data starts at offset 2.
#else // If not defined SLAVE_TEST
#if (SPI_APP_X_WIRE == SPI_APP_4_WIRE)
    #if (SPI_APP_TRANS_MODE == SPI_APP_FULL_DUPLEX)
        /* SPI send and receive in 4-wire master mode. */
        SPI_TransmitReceive((uint8_t *)&m_au8SpiTxBuf[0u], \
                            (uint8_t *)&m_au8SpiRxBuf[0u], \
                            SPI_BUFFER_LENGTH);
    #else
        /* SPI send only in 4-wire master mode. */
        SPI_Transmit((uint8_t *)&m_au8SpiTxBuf[0u], SPI_BUFFER_LENGTH);
    #endif
#else
    #ifndef SPI_APP_CUSTOM_NSS
        /* SPI send and receive in 3-wire master mode, with custom NSS pin. */
        SpiTransmitData((uint8_t *)&m_au8SpiTxBuf[0u], SPI_BUFFER_LENGTH);
        SpiReceiveData((uint8_t *)&m_au8SpiRxBuf[0u], SPI_BUFFER_LENGTH);
        SpiTransmitReceiveData((uint8_t *)&m_au8SpiTxBuf[0u], \
                               (uint8_t *)&m_au8SpiRxBuf[0u], \
                               SPI_BUFFER_LENGTH, \
                               SPI_BUFFER_LENGTH);
    #else
        #if (SPI_APP_TRANS_MODE == SPI_APP_FULL_DUPLEX)
            /* SPI send and receive in 3-wire master mode. */
            SPI_TransmitReceive((uint8_t *)&m_au8SpiTxBuf[0u], \
                                (uint8_t *)&m_au8SpiRxBuf[0u], \
                                SPI_BUFFER_LENGTH);
        #else
            /* SPI send only in 3-wire master mode. */
            SPI_Transmit((uint8_t *)&m_au8SpiTxBuf[0u], SPI_BUFFER_LENGTH);
        #endif
    #endif // #ifndef SPI_APP_CUSTOM_NSS
#endif
#endif // #ifndef SLAVE_TEST
}
}

```

程序清单 4-1 SPI 例程基本结构

说明:

1. SystemClockConfig(): 配置系统时钟, 可选。在例程 spi\_master\_base 中, 将系统时钟配置为 HRC32MHz, 如程序清单 4-2。

```

static void SystemClockConfig(void)
{
    /* Set EFM read latency when system clock greater than 24MHz. */
    EFM_SetLatency(EFM_LATENCY_1);

    /* Configure the system clock to HRC32MHz. */
    CLK_HRCInit(CLK_HRC_ON, CLK_HRCFREQ_32);
}

```

程序清单 4-2 系统时钟配置

2. SpiConfig(), SPI 配置, 包括 SPI 模块初始化, SPI 引脚初始化和 SPI 功能使能, 如程序清单 4-3。

```
static void SpiConfig(void)
{
    stc_spi_init_t stcInit;

    /* Set a default value. */
    SPI_StructInit(&stcInit);

    /* User configuration value. */
    stcInit.u32WireMode      = SPI_WIRE_MODE;
    stcInit.u32TransMode     = SPI_TRANS_MODE;
    stcInit.u32NssActiveLevel = SPI_NSS_ACTIVE;
    stcInit.u32SpiMode       = SPI_SPI_MODE;
    stcInit.u32BaudRatePrescaler = SPI_BR_DIV;

    /* The SPI register can be written only after the SPI peripheral is enabled. */
    CLK_FcgPeriphClockCmd(CLK_FCG_SPI, Enable);

    /* Initializes SPI. */
    SPI_Init(&stcInit);

    /* Set the pins to SPI function. */
    #if (SPI_APP_X_WIRE == SPI_APP_4_WIRE)
        GPIO_SetFunc(SPI_NSS_PORT, SPI_NSS_PIN, GPIO_FUNC_7_SPI);
    #elif defined SPI_APP_CUSTOM_NSS
        SpiInitNssPin();
    #endif // #if (SPI_APP_X_WIRE == SPI_APP_4_WIRE)
        GPIO_SetFunc(SPI_SCK_PORT, SPI_SCK_PIN, GPIO_FUNC_7_SPI);
        GPIO_SetFunc(SPI_MOSI_PORT, SPI_MOSI_PIN, GPIO_FUNC_7_SPI);
        GPIO_SetFunc(SPI_MISO_PORT, SPI_MISO_PIN, GPIO_FUNC_7_SPI);

    /* Enable SPI function. */
    SPI_FunctionCmd(Enable);
}
```

程序清单 4-3 SPI 配置

## 4.2 从机模式

从机模式，波特率最大支持 PCLK/6。

### 4.2.1 从机 4 线模式

该模式下，如果 SPI 检测到 SS0 输入信号变为有效电平，就需要开始向 MISO 输出信号驱动有效数据。因此，在 CPHA 位为 0 时，将 SS0 输入信号电平从无效变为有效视为开始串行传送的触发信号。当 CPHA 位为“1”时，如果在 SS0 输入信号为有效电平的状态下 SPI 检测到最初的 SCK 边沿，就需要开始向 MISO 输出信号驱动有效数据。因此，在 CPHA 位为“1”时，将 SS0 信号处于有效电平状态下的首个 SCK 边沿视为开始串行传送的触发信号。

### 4.2.2 从机 3 线模式

当 SPI 控制寄存器 SPI\_CR1 中的 SPIMDS 位为 1 时，SPI 处于时钟同步运行模式。在该模式动作时，SPI 只使用 SCK、MOSI 和 MISO 这 3 个管脚进行通信，SS0 管脚被释放可用于普通 I/O 功能。尽管时钟同步运行模式时不使用 SS0 管脚，但模块内部的运行和 SPI 运行模式是相同的。但由于没有了 SS0 的输入，所以检测不到模式故障错误。

由于时钟同步运行模式下不使用 SS0 管脚，因此，在 CPHA 位为 0 时无法进行正常通信。当 CPHA 位为“1”时，如果在 SS0 输入信号为有效电平的状态下 SPI 检测到最初的 SCK 边沿，就需要开始向 MISO 输出信号驱动有效数据。因此，在 CPHA 位为“1”时，将 SS0 信号处于有效电平状态下的首个 SCK 边沿视为开始串行传送的触发信号。所以，从机 3 线模式，只能工作在 SPI 的模式 1 和模式 3。

### 4.2.3 程序示例

例程 spi\_slave\_interrupt 展示了 SPI 从机模式配置和用法，例程 spi\_master\_base 使能宏定义 SLAVE\_TEST，即可和从机实现主从通信。这里主要介绍从机的关键代码。

主函数如程序清单 4-4 所示。

```
int32_t main(void)
{
    /* The maximum transfer rate of the slave is PCLK/6.
       The SPI clock of the slave is from the master, so the SCK frequency
       of the master SPI is at most PCLK/6. */

    /* Configure the system clock to HRC32MHz. */
    SystemClockConfig();

    /* Configures SPI. */
    SpiConfig();
}
```

```

/***** Configuration end, application start *****/

while (1u)
{
    if (m_u8RxStart)
    {
        if (++m_u32RxIdle >= SPI_IDLE_TIME)
        {
            if (m_au8SpiRxBuf[0u] == SPI_WRITE_SLAVE)
            {
                // TODO: Use the data from the master.
            }

            if (m_au8SpiRxBuf[0u] == SPI_READ_SLAVE)
            {
                // TODO: Prepare data that needs to be sent to the master.
                m_au8SpiTxBuf[0u]++;
                m_au8SpiTxBuf[1u]++;
                m_au8SpiTxBuf[2u]++;
                m_au8SpiTxBuf[3u]++;
                m_au8SpiTxBuf[4u]++;
                m_au8SpiTxBuf[5u]++;
            }
            m_u32RxIdle = 0u;
            m_u8RxStart = 0u;
            m_u8RxDataCount = 0u;
            m_u8TxDataCount = 0u;
        }
    }
}

```

程序清单 4-4 SPI 从机主函数示例代码

说明：

1. 函数 SystemClockConfig() 与 spi\_master\_base 中的同名函数一样；函数 SpiConfig() 与 spi\_master\_base 中的同名函数大同小异，一个主机一个从机。
2. 变量 m\_u8RxStart，用于表示开始接收来自主机的数据，结合变量 m\_u32RxIdle（用于统计空闲接收到最后一个数据的空闲时间），来判断收包结束。在实际应用中，可用定时器来统计空闲时间。
3. SPI\_WRITE\_SLAVE 和 SPI\_READ\_SLAVE 是自定义的两个常量，简单表示来自主机的命令类型。
4. 变量 m\_u8RxDataCount 是接收数据统计，变量 m\_u8TxDataCount 是发送数据统计。
5. 从机发送给主机的数据，必须在发送缓存为空的中断（以下用 TX\_EMPTY 表示）回调函数里填充；接收主机的数据，必须在接收缓存满的中断（以下用 RX\_FULL）里。  
TX\_EMPTY 中断产生后，会写入一个数据到数据寄存器，当数据写入数据寄存器后，数



据被移至移位寄存器（等待发送给主机），此时数据寄存器为空，然后会再次产生一次 TX\_EMPTY 中断，进入中断后，依然继续将数据写入数据寄存器，但此时，移位寄存器已经有数据，所以数据不会被移至移位寄存器，因此 TX\_EMPTY 中断不会再产生。综上所述，主机收到从机的前两个数据是无效的，有效数据从接收数据的偏移位置 2 开始，假设主机接收数组大小为 N，那么有效数据长度是 N-2；从机发送给主机的数据数组大小为 N，有效数据偏移位置从 0 开始，有效长度为 N-2。

## 5 总结

本应用笔记简要介绍了 HC32M120 系列 SPI 模块的各种功能和用法，并给出了 SPI 模块应用的基本流程，在实际开发中，用户可根据具体应用场景按需配置和应用 SPI 模块。

## 6 版本信息 & 联系方式

日期	版本	修改记录
2019/10/31	Rev1.0	初版发布



---

如果您在购买与使用过程中有任何意见或建议，请随时与我们联系。

Email: [mcu@hdsc.com.cn](mailto:mcu@hdsc.com.cn)

网址: [www.hdsc.com.cn](http://www.hdsc.com.cn)

通信地址: 上海市浦东新区中科路 1867 号 A 座 10 层

邮编: 201203

---

